Nicholas Vickery

[redacted]

7/25/2023

Summary

I know it’s in MLA format, but I do it this way so its easier on me to list out the game in its best form.

In this game, you can step into the shoes of a knight. Your mission? To slay monsters, aiming ultimately to topple the monster that reigns supreme. Right now, the gameplay revolves around annihilating monsters and bosses, aiming to surpass your own high score and striving to ascend to the highest level without being vanquished.

Gameplay is facilitated solely through keyboard commands: 'A' sends you left, 'D' sends you right, 'S' allows you to defend, 'W' is for attacking, and 'P' is reserved for using potions you gather along your journey. At this stage, your adversaries are the 'beholders' and a boss represented as a 'blue dragon'. I plan to incorporate detailed phase descriptions to portray the game's progression later in the summary, but it's key to note that the game is currently in a testing phase and is subject to enhancements and corrections.

Despite its small size and lower health (6 health points), the beholder can move slightly quicker than the boss. The boss, however, has more health (20 health points), and while its movements are slower, its attacks pack a more powerful punch. The knight, controlled by the player, has 100 health points and can outpace the monsters. Unique to the knight is the ability to heal, a feature attributable to the knight's status as a living, intelligent being – in other words, the player.

I have also started a two-player version of this game. However, testing on this version is not yet underway as I've chosen to pause its development and focus more on refining the current game.

In the HTML file, an audio element has been implemented to provide background music during gameplay, enhancing the combat experience. For now, the repertoire is limited to a single song. I've designated a 'div' element to control the player's knight character and another to manage the health system, complete with a dynamic progress bar that fluctuates based on the knight's actions, such as taking damage from enemies or consuming a healing potion.

Another significant 'div' element, identified by the id 'content', accommodates additional divisions, like 'header-text' and 'instructions'. These elements guide players by displaying the game controls. Crucial game information is displayed through two other divisions, labeled as 'level display' and 'high score display'. To enhance the user's gameplay experience, an additional coding layer includes a 'span' element that progressively updates the level display as the player advances. The potion, an essential element of the game, is depicted via a separate 'div' with its visuals and dimensions detailed in the CSS file and its properties coded in JavaScript.

In the CSS file, every division from the HTML file is given a unique aesthetic setup, defining elements like background images, character styles, and the placement of the knight in the game. Each CSS section plays a pivotal role in the game, establishing parameters for the game's elements like starting locations, instruction placements, and level systems.

Moving onto the JavaScript file, an audio function was integrated post-game testing, addressing an issue of eeriness due to the lack of background music. This function, although it comprises just two lines in JavaScript, relies heavily on the DOM to extract the 'audio' element from HTML and play the music when the player presses the 'D' key.

The variables 'knight' and 'potion' are frequently referred to in the code, performing various tasks like managing the knight's stance, spawning the potion at random locations, and assigning images to these entities. The code's action section determines the properties of the knight's sprite sheet, including the individual frames within the sheet, their animation speed, and the dimensions of each frame. Any action requiring the player to flip results in alternative animations, distinguished by the term 'flipped'.

The 'setAnimation' function dynamically alters the animations based on the knight's current action. It adjusts the knight's dimensions and image to match the ongoing action, like transitioning from idle to attack frames. The 'updateHighScore' function continually refreshes the high score, tracking the highest level completed before the player's character dies. In essence, it verifies if the current level exceeds the previously saved high score stored in local storage, ensuring the high score persists across gaming sessions.

The 'takeDamage' function diminishes the player's health by 25 points when a monster attacks and updates the progress bar accordingly. If the player's health drops to zero, the page reloads, and the health bar resets to 100. The 'usePotion' function, once invoked, checks for the potion's availability. If the player has a potion, it grants them an additional 50 health points and updates the progress bar but resets the potion availability to false until the next pick-up.

The 'spawnEnemy' and 'spawnBoss' functions essentially perform the same task, spawning enemies and bosses respectively, and adding them to the gameplay with unique health, speed, and direction parameters. These spawns are regulated based on the current game level and the number of existing entities to maintain balance.

The 'gameLoop' function constitutes the main loop of the game, updating the game state according to each frame of the knight and the enemies. It computes the Delta time, updates animations, verifies enemy or boss kill status, and refreshes the level if cleared. This function, along with the collision system, required a significant amount of time and attention due to their complexity.

Similarly, the 'update' function alters the game state for each frame. It computes the current frame index for animations, mobilizes the knight for running actions, detects collisions between the player and enemies, and processes the outcomes. For clarity, I've dubbed this system the 'collision system'.

Event listeners have been integrated to bind specific keys ('W', 'S', 'D', 'A', 'P') to in-game actions. These listeners work in sync with key up and key down events. Key up events revert the knight to the 'idle' state upon releasing a key, whereas key down events trigger corresponding actions based on the pressed key.

While I've incorporated specific function calls, I've also ensured that the code includes detailed comments. This makes the code readable and self-explanatory, for both me and anyone else reviewing it, and allows for easy future updates for the game.

Phases:

Phases	
Phase 1:	• Make base game with functions for player and enemies. • Set foundation for player functionality. • Set foundation for enemy functionality. • Introduce first boss. • Add potion and potion functionality. • Add in high score and levels. • Add in music related to the lone knight
Phase 2:	• Add in functionality for enemies dying. • Add in replacement photo for enemies dying. • Add in more versions of enemies. • Add more intelligent bosses (run away, steal potion, etc.) • Readjust hitbox parameters. • Add in another boss. • Add in new area the player can run to. • Add more music to playlist
Phase 3:	To be determined
Phase 4:	To be determined